

# **Developer Manual for Southwestern University Course Scheduler Service**

Computer Science Capstone '24

Yahya Hamdallah, Mark Mueller, Kate Nguyen, Colby Sullivan

Instructor: Dr. Barbara Anthony

<b>Integrated Development Environment (IDE)</b>	<b>3</b>
Installing Visual Studio Code	3
<b>Workflow Control</b>	<b>3</b>
Installing GitHub Desktop	3
Using GitHub Desktop	3
<b>Hosting: Python Anywhere</b>	<b>3</b>
Setting Up Python Anywhere	3
Creating the Virtual Environment	4
<b>Languages and Frameworks: Python, HTML, CSS, JavaScript, Flask</b>	<b>6</b>
Python	6
Installing Python	6
HTML	6
CSS	7
JavaScript	7
Flask	7
Installing Flask	7
Live View	7
<b>GNU Linear Programming Kit (PyGLPK)</b>	<b>8</b>
<b>Comma Separated Values (CSV)</b>	<b>9</b>
<b>Editing CSVs</b>	<b>10</b>
<b>Making CSVs</b>	<b>10</b>
Output Explanation	11
<b>Database</b>	<b>11</b>
Google Firebase	11
Autofill Example	12
<b>File Directory</b>	<b>13</b>
Navigating Files	13
Frontend Files	13
Backend Files	13
<b>Website Maintenance</b>	<b>14</b>
Website Down	14
Pushing Changes	14
<b>Future/On-Going Work</b>	<b>15</b>

## Integrated Development Environment (IDE)

The IDE used was [Visual Studio Code](#). This isn't necessary, but it pairs well with Github.

### Installing Visual Studio Code

1. Go to Visual Studio Code [Homepage](#)
2. Click [Downloads](#)
3. Download Visual Studio Code
4. Follow the instructions on the installer

## Workflow Control

The primary workflow control we used was [GitHub Desktop](#). Our code is hosted on Github, so GitHub Desktop or other programs, like GitKraken are needed to import the repo.

### Installing GitHub Desktop

1. Click "Download for ..."
2. Follow the instructions on the installer

### Using GitHub Desktop

1. Clone Repository of Course Scheduler
2. Create your own Branch
3. Pull from Main to your branch
4. Open in Visual Studio Code

## Hosting: [Python Anywhere](#)

### Setting Up Python Anywhere

Our project uses the free version of Python Anywhere to host the site.

1. Go to Python Anywhere [Homepage](#)
2. Create a Beginner Account
3. Go to the Web tab
4. Either create your domain or use the predetermined one
5. Click Next
6. Select Flask as your Python Web framework
7. Click Next
8. Python 3.10 (or up to discretion)
9. Click Next
10. Click Next again on the Quickstart new Flask project
11. Your web application has been deployed.

### Initial setup for PythonAnywhere

After creating a web application in Python 3.10, go to the console page and run a bash console using the following commands.

### Creating the Virtual Environment

Unset

```
mkvirtualenv myvirtualenv --python=/usr/bin/python3.10
```

To test if it worked, run

Unset

```
which python
```

To activate and add modules to environment

Unset

```
workon myvirtualenv
```

Installing the necessary modules

Unset

```
pip install glpk  
pip install Flask
```

When you are done adding to your virtual environment, deactivate it so we can clone our repository.

Unset

```
deactivate
```

Cloning the repository and setting up the directories.

Unset

```
git clone -b main
https://github.com/MuellMark/Course-Scheduler.git
```

You can also substitute main for any other branches.

[Go to the Web section.](#)

Go to the web directory and make it look like this.

The path to Virtualenv, the Working directory, and the Source code should be similar, if not exactly the same.

Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. [More info here](#). You need to **Reload your web app** to activate it; NB - will do nothing if the virtualenv does not exist.

</home/colbySullivan/.virtualenvs/flaskenv>

[Start a console in this virtualenv](#)

Code:

What your site is running.

Source code:	<a href="/home/colbySullivan/Course-Scheduler/src">/home/colbySullivan/Course-Scheduler/src</a>	<a href="#">Go to directory</a>
Working directory:	<a href="/home/colbySullivan/Course-Scheduler/src">/home/colbySullivan/Course-Scheduler/src</a>	<a href="#">Go to directory</a>
WSGI configuration file:	<a href="/var/www/colbysullivan_pythonanywhere_com_wsgi.py">/var/www/colbysullivan_pythonanywhere_com_wsgi.py</a>	
Python version:	3.10 <a href="#">✎</a>	

Run these commands if you have issues pulling new code from the repository.

Unset

```
cd /Course-Scheduler
git pull
```

If there are merge conflicts with the files currently in the web application, run this command. Be aware that this will remove any changes you make locally in PythonAnywhere.

Unset

```
git reset --hard
```

**Languages and Frameworks:** [Python](#), [HTML](#), [CSS](#), [JavaScript](#), [Flask](#)

## Python

Python is an interpreted language that does not require installation. This was used for backend development, both to create the API and for the algorithm.

### Installing Python

#### **Windows:**

1. The easiest way to install Python will be through the Windows App Store
2. Download Python from this [link](#)
3. Restart your device after the download is complete, and reopen the project

#### **Mac:**

##### A.

1. Download Python from this [page](#)
2. Go through the installer steps
3. Verify it is installed by searching for Python in your applications

##### B.

1. Install homebrew from this [page](#)
2. Run the command  

```
brew install python@[version]
```
3. Verify it is installed by running  

```
python --version
```

## HTML

HyperText Markup Language (HTML) does not need installation. It is a standard language used to create and design the webpage.

## CSS

Cascading Styling Sheets (CSS) does not need installation. They allow us to control the structure and appearance of the web page with more flexibility regarding colors, fonts, and layout.

## JavaScript

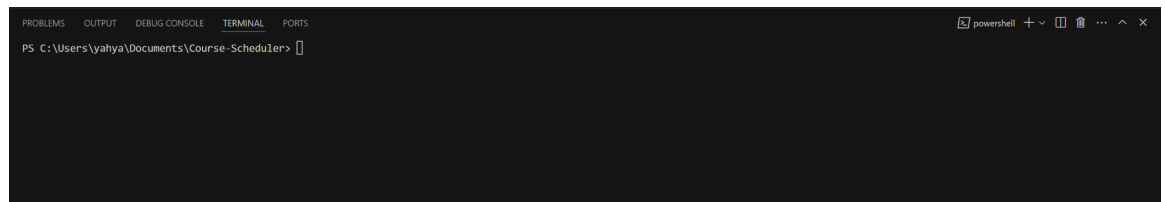
JavaScript does not need installation. It allows for dynamic features on webpages and was used for more complex features and our animations.

## Flask

Flask is a flexible web framework that requires installation. It is designed for Python and allows for a simple and quick website setup.

## Installing Flask

1. Open a PowerShell terminal in Visual Studio

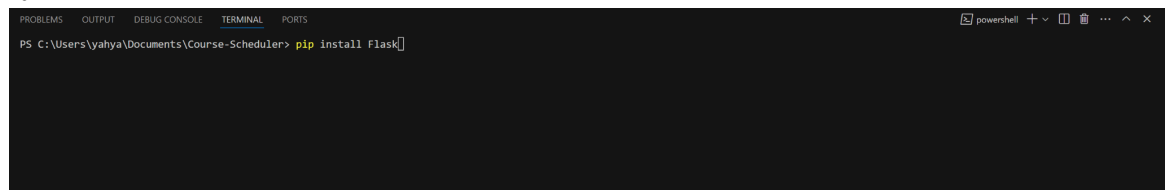


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\yahya\Documents\Course-Scheduler>

```

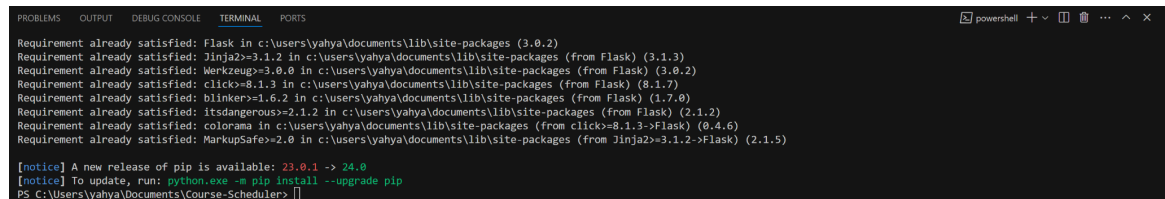
2. Type in - pip install Flask



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\yahya\Documents\Course-Scheduler> pip install Flask

```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Requirement already satisfied: Flask in c:\users\yahya\documents\lib\site-packages (3.0.2)
Requirement already satisfied: Jinja2>=3.1.2 in c:\users\yahya\documents\lib\site-packages (from Flask) (3.1.3)
Requirement already satisfied: Werkzeug>=3.0.0 in c:\users\yahya\documents\lib\site-packages (from Flask) (3.0.2)
Requirement already satisfied: click>=8.1.3 in c:\users\yahya\documents\lib\site-packages (from Flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in c:\users\yahya\documents\lib\site-packages (from Flask) (1.7.0)
Requirement already satisfied: itsdangerous>=2.1.2 in c:\users\yahya\documents\lib\site-packages (from Flask) (2.1.2)
Requirement already satisfied: colorama in c:\users\yahya\documents\lib\site-packages (from click>=8.1.3->Flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\yahya\documents\lib\site-packages (from Jinja2>=3.1.2->Flask) (2.1.5)

[notice] A new release of pip is available: 23.0.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\yahya\Documents\Course-Scheduler>

```

## Live View

This is the easiest way for developers to see changes in real time before deploying to Python Anywhere.

## 1. Open a bash terminal in Visual Studio

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
yahya@YahyaHLaptop MINGW64 ~/Documents/Course-Scheduler (YahyaNew)
$

```

## 2. cd src

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
yahya@YahyaHLaptop MINGW64 ~/Documents/Course-Scheduler (YahyaNew)
$ cd src
yahya@YahyaHLaptop MINGW64 ~/Documents/Course-Scheduler/src (YahyaNew)
$

```

## 3. python app.py

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
yahya@YahyaHLaptop MINGW64 ~/Documents/Course-Scheduler/src (YahyaNew)
$ python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 112-475-209

```

## 4. Click on HTTP link and it will open in a browser window

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
yahya@YahyaHLaptop MINGW64 ~/Documents/Course-Scheduler/src (YahyaNew)
$ python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 112-475-209

```

## 5. When you want to see the edits you made on the webpage, refresh the page; the changes will be shown.

**GNU Linear Programming Kit (PyGLPK)**

The Scheduler operates from a Python script running the Python GNU Linear Programming Kit (PyGLPK). The script is fed into 1 CSV file containing all the course and faculty constraints. A linear program is created and modeled based on these constraints as a 2D matrix. This linear program is then passed into PyGLPK, where it is solved, checked for feasibility, and exported to a separate CSV file. This CSV file has been passed back to the website for proper display. If you plan to use this module outside of PythonAnywhere, be aware that it will only work locally through Mac if you use homebrew. If you decide to do that, there is no homebrew virtual environment, so you cannot host a locally run server unless you use an Ubuntu Linux server. To get PyGLPK running on Linux, simply install the apt-get command on your local machine.



## Comma Separated Values (CSV)

Comma-separated values (CSV) is a text file format that stores tabular data in regular text, using commas to separate values. Each line in the file represents one data record in the table. These files are a large component of the Southwestern Course Scheduler application. They define the course and faculty restrictions in the linear program to generate the course schedule.

```

<forced_courses>
CS21,m200

<course_restrict>
CS11,TRUE,CS21,EXP1,ORG1,ALG1,m330,$
CS21,TRUE,CS11,EXP1,ORG1,ALG1,m330,$
DIS1,FALSE,ORG1,PRL1,ARI1,TOC1,DAT1,OPR1,NUA1,m330,$
ORG1,TRUE,DIS,PRL1,ARI1,TOC1,DAT1,OPR1,NUA1,CS11,CS21,EXP1,ALG1,m330,$
PRL1,FALSE,DIS,ORG1,ARI1,TOC1,DAT1,OPR1,NUA1,m330,$
ALG1,FALSE,SYS1,CAP1,ARI1,TOC1,DAT1,OPR1,NUA1,CS11,CS21,EXP1,ORG1,m330,$
SYS1,FALSE,ALG1,CAP1,ARI1,TOC1,DAT1,OPR1,NUA1,m330,$
CAP1,TRUE,ALG1,SYS1,ARI1,TOC1,DAT1,OPR1,NUA1,m330,$
ARI1,FALSE,TOC1,DAT1,OPR1,NUA1,DIS1,ORG1,PRL1,ALG1,SYS1,CAP1,m330,$
TOC1,FALSE,ARI1,DAT1,OPR1,NUA1,DIS1,ORG1,PRL1,ALG1,SYS1,CAP1,m330,$
DAT1,FALSE,ARI1,TOC1,OPR1,NUA1,DIS1,ORG1,PRL1,ALG1,SYS1,CAP1,m330,$
OPR1,FALSE,ARI1,TOC1,DAT1,NUA1,DIS1,ORG1,PRL1,ALG1,SYS1,CAP1,m330,$
NUA1,FALSE,ARI1,TOC1,DAT1,OPR1,DIS1,ORG1,PRL1,ALG1,SYS1,CAP1,m330,$
EXP1,FALSE,CS11,CS21,ORG1,ALG1,$

<faculty_restrict>
Prof_1,TRUE,ALG1,CAP1,PRL1,OPR1,EXP1,m200,$
Prof_2,FALSE,CS11,TOC1,$
Prof_3,FALSE,DIS1,$
Prof_4,FALSE,NUA1,$
Prof_5,FALSE,CS21,ORG1,DAT1,SYS1,ARI1,$

```

```

<swapped_courses>
CS21,CS22
<forced_courses>
NUA1,m930
<course_restrict>
CS21,TRUE,m800,m330,NUA1,$,CS211
CS22,TRUE,m800,m330,NUA1,$,CS211
NUA1,TRUE,m800,CS21,CS22,$,NUA11

<faculty_restrict>
Faculty 1,TRUE,CS21,NUA1,$
Faculty 2,FALSE,CS22,$

```

These are 2 examples of the CSVs passed through to the Python Script. The first is a more realistic example of a schedule and its constraints. The second demonstrates the possible tags and will be used to explain how they function.

- **<swapped\_courses>**: Also known as the swap tag, it indicates when 2 courses are to be swapped. This functions differently than the other tags, as they only appear after a given schedule is created. This is because times must be set for two courses to be swapped. The two courses, CS21 and CS22, are the ones that are to be swapped. Internally, this works by getting the times each course would be taught and then making 2 forced tags where their times are swapped.
- **<forced\_courses>** Also known as the force tag, it indicates when a course must be completed at a certain time. Like the swapped tag, this is optional, as some schedules will not have any forced times. These tags have the course and then the time they must be held- in this case, NUA1 must be held at m930. There is code in place to ensure only one tag is allowed for each class, and when a new one is indicated, it replaces the one already in the file, if applicable.
- **<course\_restrict>**: Also known as the course tag. This is one of the 2 required tags, as course restrictions are necessary to create a schedule. Each line follows a format: first, the course name + section number are listed, followed by a True/False to indicate whether or not that course is a 4-contact hour course. Any courses listed after that are courses that cannot be taken at the same time. Any times listed indicate times the

course cannot be held. The \$ indicates the end of the file for the Python script, and any text listed after is reserved for the course name.

- <faculty\_restrict>: Also known as the faculty tag. This is the other required tag, as faculty restrictions are also necessary to create a schedule. These follow a similar format to the course tag rows. First is the faculty's name, then a boolean indicating whether or not they need to teach outside of prime time. Any courses listed are the courses the faculty member is teaching, and any times indicate times they cannot teach.

The CSV file parsing is generally resilient, so extra newlines and the order of the courses/times don't matter. However, the name and the boolean value must be first for each line, and the \$ needs to be last or second to last.

### Editing CSVs

Swaps and forcing times can be done on the display page. After every change, the schedule is checked for feasibility. If it is infeasible, the site will indicate this and not change the schedule. However, there is currently no way of deleting the added forced tags other than overwriting them at a different time. To delete one, download the file and open it in an editor (text edit, sheets, and excel, for example). Remove the line with course forced at a time and run the program again.

### Making CSVs

CSVs are difficult to create manually, so we offer a tool for users to create them on the CSV creation page:

**Course Table** Save as CSV Next

Clear Table Add Row New Course

	Class name	Abbreviation	4 Contact Hours	Sections	Unavailable Times	Select CourseID	
-	Computer Science 1	CS1	Yes	3	MWF <input type="checkbox"/> 8:00 <input type="checkbox"/> 9:30 <input type="checkbox"/> 11:00 <input type="checkbox"/> 2:00 <input type="checkbox"/> 3:30 TTh <input type="checkbox"/> 8:30 <input type="checkbox"/> 10:00 <input type="checkbox"/> 11:30 <input checked="" type="checkbox"/> 1:00 <input type="checkbox"/> 2:30	CS13	Add Conflicting Course

**Faculty Table**

Clear Table Add Row

	Professor Name	Prime time	Classes	Unavailable Times	
-	Barbra Anthony	Yes	CS13	MWF <input checked="" type="checkbox"/> 8:00 <input type="checkbox"/> 9:30 <input type="checkbox"/> 11:00 <input type="checkbox"/> 2:00 <input type="checkbox"/> 3:30 TTh <input type="checkbox"/> 8:30 <input type="checkbox"/> 10:00 <input type="checkbox"/> 11:30 <input type="checkbox"/> 1:00 <input type="checkbox"/> 2:30	Add Course Taught

Output:

<course-table>

CS1,TRUE,3,t100,CS13,\$,Computer Science 1

<faculty-table>

Barbra Anthony,TRUE,CS13,m800,\$

Output Explanation

<course-table>						
Abbreviation	4 Contact Hour	Sections	Unavailable Times	Course ID	End of File	Class Name
CS1,	True,	3,	t100	CS13,	,\$,	Computer Science 1
<faculty-table>						
Professor	Prime Time	Courses Taught	Unavailable Time			
Barbra Anthony,	TRUE,	CS13,	m800			

These generated schedules work a bit differently. In the third slot of the course tag rows, there is a number indicating the number of sections of a given course. The script expands this number to look like the schedule examples above. These only occur when generated directly from the created CSV page; when downloaded again, they are converted to another form.

**Database**

Google Firebase

Our application integrates a Google Firebase database to store essential course data, which is particularly utilized in the "Create CSV" page. Information like the course name, abbreviation, number of sections, and course ID are recorded when users enter courses into the database. By using an autofill feature in the course table, this configuration not only saves the data for future accessibility but also improves user productivity. In particular, the process is streamlined by automatically filling in static information, such as contact hours and abbreviations for courses taught frequently. As a result, faculty members can streamline their workflow by only having to update semester-specific dynamic information, like unavailable times and conflicting courses.

```

temp_courses
├── CS1
│   ├── abbreviation: "CS1"
│   ├── class_name: "Computer Science 2"
│   ├── contact_hours: "TRUE"
│   └── sections: "1"
├── CS12
│   ├── abbreviation: "CS1"
│   ├── class_name: "Computer Science 1"
│   ├── contact_hours: "FALSE"
│   └── sections: "2"
└── CS2
    ├── abbreviation: "CS1"
    ├── class_name: "Computer Science 1"
    ├── contact_hours: "FALSE"
    └── sections: "2"
    
```

Database Entry Example

Autofill Example

**Course Table** Save as CSV Next

Clear Table Add to Firebase Add Row New Course

Class name	Abbreviation	Contact Hours	Sections	Unavailable Times	Select CourseID
<input type="text" value="Enter New Course"/>	<input type="text" value="Enter Course Abbreviation"/>	<input type="text" value="No"/>	<input type="text" value="Number of Sections"/>	MWF <input type="checkbox"/> 8:00 <input type="checkbox"/> 9:30 <input type="checkbox"/> 11:00 <input type="checkbox"/> 2:00 <input type="checkbox"/> 3:30 TTh <input type="checkbox"/> 8:30 <input type="checkbox"/> 10:00 <input type="checkbox"/> 11:30 <input type="checkbox"/> 1:00 <input type="checkbox"/> 2:30	<input type="text" value="Enter Course ID"/>

Add Conflicting Course

**Course Table** Save as CSV Next

Clear Table Add to Firebase Add Row New Course

Class name	Abbreviation	Contact Hours	Sections	Unavailable Times	Select CourseID
<input type="text" value="Computer Science 1"/>	<input type="text" value="CS1"/>	<input type="text" value="4"/> Yes	<input type="text" value="1"/>	MWF <input type="checkbox"/> 8:00 <input type="checkbox"/> 9:30 <input type="checkbox"/> 11:00 <input type="checkbox"/> 2:00 <input type="checkbox"/> 3:30 TTh <input type="checkbox"/> 8:30 <input type="checkbox"/> 10:00 <input type="checkbox"/> 11:30 <input type="checkbox"/> 1:00 <input type="checkbox"/> 2:30	<input type="text" value="CS11"/>

Add Conflicting Course

## File Directory

<pre>   ✓ COURSE-SCHEDULER   &gt; .github   &gt; .venv2   &gt; Legacy-Code   ✓ src   &gt; __pycache__   &gt; .venv   &gt; CSV_Files   &gt; CSV-Files   &gt; not_in_use_templates   &gt; pyscript-main   &gt; Python_Code   &gt; static   ✓ templates   🐘 about-howto.php   🐘 copyofdisplay.php   🐘 csv_option.php   🐘 display.php   🐘 dynamic_merge.php   🐘 faq.php   🐘 final_schedule_result.php   🐘 import_csv.php   🐘 landing_page.php   ☰ =   🔄 app.py   🔄 File_Convertor.py   ☰ howtorun.txt   📄 input.csv   ☰ newfile.txt   🔄 originalPyGLPKGenerate.py   🔄 PyGLPK_Solver.py   📄 swap.csv   🔄 test.py   📄 .gitignore   📄 output.csv   ⓘ README.md   📄 schedulerTest.sql   📄 user_output.csv </pre>	<h3><u>Navigating Files</u></h3> <p>Flask requires that specific files be in special file paths. For CSS files they, must be in src/static/CSS. For HTML and php files, they must be in src/templates.</p> <h3><u>Frontend Files</u></h3> <p>These files will be under src -&gt; templates</p> <ul style="list-style-type: none"> <li>• about-howto.php</li> <li>• copyofdisplay.php</li> <li>• csv-options.php</li> <li>• display.php</li> <li>• dynamic_merge.php</li> <li>• faq.php</li> <li>• final_scheldule_result.php</li> <li>• import_csv.php</li> <li>• landing_page.php</li> </ul> <h3><u>Backend Files</u></h3> <p>These files will be under src</p> <ul style="list-style-type: none"> <li>• app.py</li> <li>• File_Convertor.py</li> <li>• originalPyGLPKGenerate.py</li> <li>• PyGLPK_solver.py</li> <li>• test.py</li> </ul>
--	--

## Website Maintenance

### Website Down

If the website is ever down or errors display the schedule, the script can still be called directly from a Python prompt. The file to call is `src/File_Convertor.py` (the whole path may be required, but it is different for each machine). There are 3-4 command line params that are needed:

1. The full path to the CSV file you're calling. This would be the file you'd upload to the upload page.
2. The type of operation you're doing on the file. Most times, this will be "csv," but "time" and "swap" are also possible options. These indicate whether a time will be forced or two courses are to be swapped.
3. The next command line param is the export type. Simply putting "both" is recommended, as it creates both the output for the site and for downloading simultaneously, going to `output.csv` and `user_output.csv`, respectively.
4. The final command line param is ONLY for swaps. You'll need to provide the file of a previous solution. This should be the full path of `output.csv`.

Altogether, the commands should look like these examples:

Generating a schedule from `test_swap.csv`:

```
<path>/src/File_Convertor.py <path>/src/CSV_Files/test_swap.csv csv both
```

Swapping two courses in `test_swap.csv`

```
<path>/src/File_Convertor.py <path>/src/CSV_Files/test_swap.csv swap both <path>/output.csv
```

### Pushing Changes

To push changes, go to PythonAnywhere, open up a bash terminal, and type in these commands.

Unset

```
cd /Course-Scheduler
git pull
```

If the commands are successful, you will see the changes pulled in, and then you can proceed to the next step. If there is an error, you simply need to reset the head of your repository clone by using the command below.

Unset

```
git reset --hard
```

Once you have successfully reloaded, go back to the web section of PythonAnywhere and click the green reload button at the top of the page.

### Future/On-Going Work

Many additions could be made in the future, such as:

- Fix the schedule layout; it's all one column, and splitting it would be useful
- Reporting on what was infeasible when swapping courses or forcing times. It only indicates if it is infeasible, not what broke the schedule.
- Replacing the effects of a forced course without manually editing the CSV file.
- The database currently contains only Math and Computer Science courses. Expanding to more departments at Southwestern University or even other universities would be great. This would likely require redoing the time columns and adding them to the database.